深入浅出Linux设备驱动之并发控制（1） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022__E6_B7_B1_E5_85_A5_E6_B5_85_E5_c103_144583.htm 在驱动程序中，当多个线程同时访问相同的资源时（驱动程序中的全局变量是一种典型的共享资源），可能会引发"竞态"，因此我们必须对共享资源进行并发控制。Linux内核中解决并发控制的最常用方法是自旋锁与信号量（绝大多数时候作为互斥锁使用）。 自旋锁与信号量"类似而不类"，类似说的是它们功能上的相似性，"不类"指代它们在本质和实现机理上完全不一样，不属于一类。 自旋锁不会引起调用者睡眠，如果自旋锁已经被别的执行单元保持，调用者就一直循环查看是否该自旋锁的保持者已经释放了锁，"自旋"就是"在原地打转"。而信号量则引起调用者睡眠，它把进程从运行队列上拖出去，除非获得锁。这就是它们的"不类"。 但是，无论是信号量，还是自旋锁，在任何时刻，最多只能有一个保持者，即在任何时刻最多只能有一个执行单元获得锁。这就是它们的"类似"。 鉴于自旋锁与信号量的上述特点，一般而言，自旋锁适合于保持时间非常短的情况，它可以在任何上下文使用；信号量适合于保持时间较长的情况，会只能在进程上下文使用。如果被保护的共享资源只在进程上下文访问，则可以以信号量来保护该共享资源，如果对共享资源的访问时间非常短，自旋锁也是好的选择。但是，如果被保护的共享资源需要在中断上下文访问（包括底半部即中断处理句柄和顶半部即软中断），就必须使用自旋锁。 与信号量相关的API主要有： 定义信号量 struct semaphore sem.初始化信号量 void sema_init (struct

semaphore *sem, int val).该函数初始化信号量，并设置信号量sem的值为val void init_MUTEX (struct semaphore *sem).